MUSIC RECOMMENDATION BASED ON MOOD

By

NISHANTH N	**20BRS1013**
SIDDHARTH IYYAPAN G	**20BEC1194**
SANTHOSH S	**20BEC1178**
BHARAT S	**20BEC1152**

A project report submitted to

SMART INTERNZ

in partial fulfilment of the requirements for the course of

ARTIFICIAL INTELLIGENCE

JULY 2023

ABSTRACT

The Mood-Based Music Recommendation System is a project aimed at providing personalized music recommendations to users based on their predicted mood. The system utilizes image-based emotion prediction, deep learning techniques, and a recommendation engine to offer a tailored music listening experience.

The project begins by collecting a labelled dataset of images representing various emotions and their respective labels. A deep learning model, ResNetVGG50, is trained on this dataset to accurately predict emotions from user-uploaded images. The predicted mood is then passed to the recommendation engine, which generates suitable music suggestions based on factors like genre, tempo, and user preferences.

To provide an interactive interface for users, a web application is developed using Flask and HTML. The application allows users to upload images, displays the predicted emotions, and showcases the recommended song.

The design of the project focuses on usability, scalability, and privacy. The user interface provides a clean and visually appealing experience, while the recommendation engine ensures accurate and personalized song suggestions.

Overall, the Mood-Based Music Recommendation System aims to enhance the user's music listening experience by leveraging deep learning, image-based emotion prediction, and personalized recommendations. It offers a platform for users to explore and discover music that resonates with their emotions and preferences.

TABLE OF CONTENTS

1.1 Objectives:

The objective of this project is to develop a mood-based music recommendation system using image-based emotion prediction. Music has a profound impact on our emotions and can greatly influence our mood. By leveraging visual cues and emotions, we aim to enhance the music recommendation process and provide a personalized and immersive music listening experience.

In today's digital age, there is an abundance of music available across various platforms. However, finding the right music that matches our mood can be a daunting task. Traditional music recommendation systems often rely on user preferences, historical data, or collaborative filtering techniques. While these approaches have proven effective to some extent, they may not capture the subtle nuances of our current emotional state.

To address this limitation, we propose a novel approach that incorporates image-based emotion prediction. We utilize the ResNetVGG50 model, a deep learning architecture renowned for its excellent performance in image recognition tasks. By fine-tuning this model on a labelled dataset of images representing different emotions, we can accurately predict the emotions associated with a given image.

1.2 Application:

Applications of a mood-based music recommendation system:-

Personalized Music Streaming Platforms:

Mood-based music recommendation systems can be integrated into popular music streaming platforms like Spotify, Apple Music, or Pandora. By understanding the user's mood, the system can generate customized playlists or suggest relevant songs to enhance the user's music listening experience.

Entertainment and Media Industry:

Mood-based music recommendation systems can be used in various entertainment and media applications. For example, they can be incorporated into movie or TV show streaming platforms to suggest soundtracks or background music that complement the emotional tone of the content.

Fitness and Well-being Apps:

Music plays a crucial role in setting the mood and motivation during workouts or relaxation sessions. Mood-based music recommendation systems can be employed in fitness and well-being apps to offer tailored music playlists that align with the user's energy level or desired ambiance.

Retail and Hospitality Environments:

Music has a significant impact on customer experience in retail stores, restaurants, or hotels. By analysing the mood of the environment or individual customers, a mood-based music recommendation system can select appropriate background music to create a specific atmosphere and enhance customer satisfaction.

Mental Health and Therapy:

Music has therapeutic effects on mental health and can be used as a tool for relaxation, mood regulation, and emotional expression. Mood-based music recommendation systems can be utilized in mental health applications, such as therapy sessions or mindfulness apps, to suggest music that supports emotional well-being and facilitates relaxation.

Event Planning and Social Gatherings:

Mood-based music recommendation systems can be employed in event planning platforms to suggest suitable playlists or music tracks for different types of events, parties, or social gatherings. This helps create the desired ambiance and cater to the emotional preferences of the attendees.

Radio and Broadcasting:

Traditional radio stations and broadcasting platforms can benefit from incorporating mood-based music recommendation systems. By understanding the mood of the target audience or the current emotional context, broadcasters can curate playlists or select songs that resonate with the listeners' emotions.

1.3 Features:

Key features that can be incorporated into a mood-based music recommendation system:

Image-based Emotion Prediction:

Utilize deep learning models such as ResNetVGG50 to predict emotions from user-uploaded images. The model is trained on a labelled dataset of images representing different emotions.

Mood Classification:

Categorize the predicted emotions into specific mood categories such as happy, sad, calm, energetic, etc. This step helps to further refine the music recommendation process.

Music Recommendation Engine:

Develop a recommendation algorithm that takes into account the predicted mood/emotion and suggests appropriate songs or playlists. Consider

factors such as genre, tempo, lyrics, and artist preferences when generating music recommendations.

Web Application Interface:

Develop a user-friendly web application using frameworks like Flask and HTML. Design an intuitive and visually appealing interface where users can upload images and view recommended songs.

2.1 Design:

The design of the mood-based music recommendation system project can be summarized as follows:

Data Collection: Gather a labelled dataset of images representing different emotions and associated songs. This dataset will be used for training the emotion prediction model.

Emotion Prediction Model: Utilize a deep learning model, such as ResNetVGG50, to train an image-based emotion prediction model. Fine-tune the model on the collected dataset to accurately classify emotions from user-uploaded images.

Web Application Development: Build a web application using Flask and HTML to create an interactive interface for users. The web app should allow users to upload images, display predicted emotions, and provide recommendations for suitable songs based on the predicted mood.

Integration with Music Recommendation Engine: Develop a music recommendation engine that takes the predicted mood/emotion as input and suggests appropriate songs or playlists. Consider factors such as genre, tempo, and user preferences in generating the recommendations.

Music Streaming Platform Integration: Integrate the recommendation system with popular music streaming platforms like Spotify. Enable users to directly listen to the recommended songs or add them to their preferred streaming service.

In summary, the design of the mood-based music recommendation system involves training an emotion prediction model, developing a web

application with a user-friendly interface, and integrating a music recommendation engine. The system aims to enhance the user's music listening experience by leveraging image-based emotion prediction and tailored song suggestions.

2.2 Concepts:

Emotion Recognition: Emotion recognition involves analysing visual cues such as facial expressions, body language, or images to determine the emotional state of an individual. Deep learning models, such as convolutional neural networks (CNNs), are often used for image-based emotion recognition.

Deep Learning Models: Deep learning models, such as ResNetVGG50, can be utilized for image classification and emotion prediction tasks. These models are trained on large datasets and can extract high-level features from images, enabling accurate emotion recognition.

Dataset Preparation: To train the emotion prediction model, you will need a labelled dataset of images with associated emotions. This dataset should be diverse and representative of different emotional states. Annotations can be done manually by human experts or through crowd-sourcing techniques.

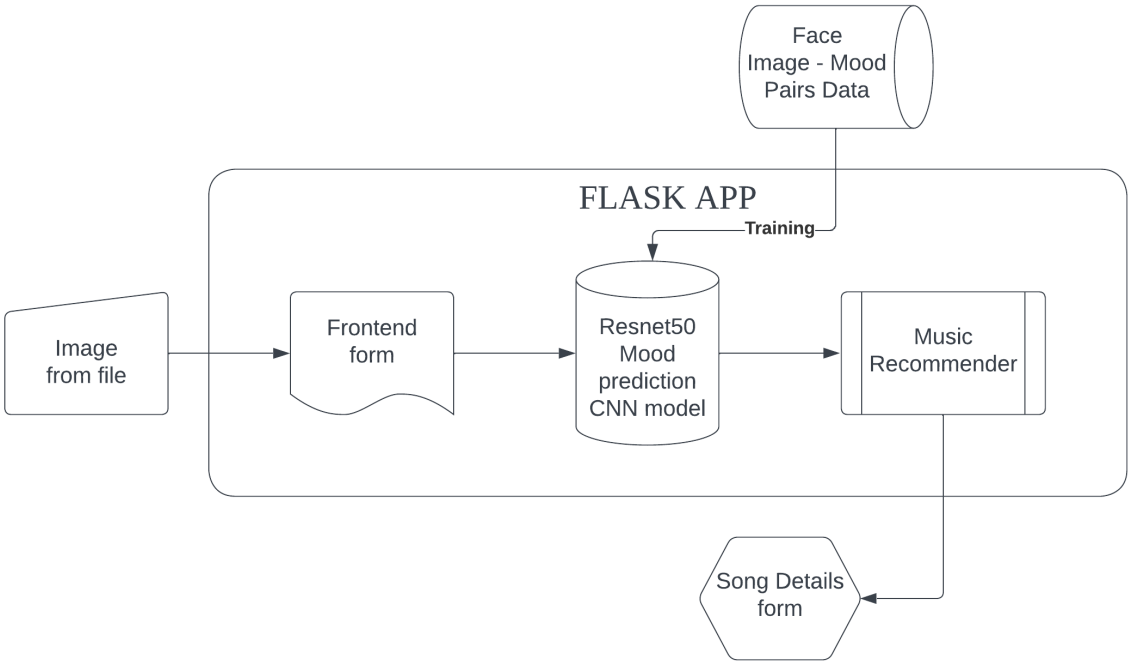
Feature Extraction: Preprocessing and feature extraction are crucial steps in image-based emotion prediction. Techniques like resizing, normalization, and data augmentation can be applied to prepare the images for input into the deep learning model. Additionally, extracting meaningful features from intermediate layers of the model can be beneficial for emotion prediction.

Music Recommendation Algorithms: Various recommendation algorithms can be employed to suggest music based on the predicted emotions. Collaborative filtering, content-based filtering, or hybrid approaches can be used to generate personalized music recommendations. These algorithms consider factors such as genre, artist, user preferences, and similarity measures.

User Feedback and Iterative Improvement: Incorporating user feedback is crucial for enhancing the recommendation system's accuracy and user satisfaction. Feedback mechanisms such as ratings, likes, or explicit feedback on recommended songs can be used to fine-tune the recommendation algorithm and improve future suggestions.

Web Application Development: Flask, a Python web framework, can be used to develop the web application for your mood-based music recommendation system. Flask enables you to handle user requests, process image uploads, and display recommended songs in a web interface. HTML and CSS are utilized for designing and styling the user interface.

2.3 Block diagram:



3. Software Implementation

3.1 Setup

Deep learning model :-

The coding for the deep learning model was done in Google Colab in python. The model was trained for 50 epochs.

Webapp:

The webapp for the project was designed using flask and html in visual studio.

There are 3 files here, app.py, index.html and display.html.

App.py is the driver code which integrates all the codes to form the webapp.

Index.html is the code for the webapp homepage where the user can upload the image of their face.

Display.html is the code for the page which displays a music recommended just for the user along with the Spotify link to that music.

Code Implementation

Deep learning model:

```
!pip install kaggle
import os

os.environ['KAGGLE_CONFIG_DIR'] = '/content'
!kaggle datasets download -d msambare/fer2013 -p /content/dataset
!unzip /content/dataset/fer2013.zip
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.style.use('default')

import os
import tensorflow as tf
import keras
import cv2

from sklearn.model_selection import train_test_split

from tensorflow.keras.preprocessing.image import ImageDataGenerator,
load_img, img_to_array
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint,
ReduceLROnPlateau
from tensorflow.keras.utils import plot_model
```

```
from tensorflow.keras import layers , models, optimizers

from tensorflow.keras.models import Sequential, Model
from tensorflow.keras.layers import *
from tensorflow.keras.applications import ResNet50V2
train_dir = '/content/train/'
test_dir = '/content/test/'

def Classes_Count( path, name):
    Classes_Dict = {}

    for Class in os.listdir(path):

        Full_Path = path + Class
        Classes_Dict[Class] = len(os.listdir(Full_Path))

    df = pd.DataFrame(Classes_Dict, index=[name])

    return df

Train_Count = Classes_Count(train_dir,
'Train').transpose().sort_values(by="Train", ascending=False)
Test_Count = Classes_Count(test_dir,
'Test').transpose().sort_values(by="Test", ascending=False)
```

```
pd.concat([Train_Count,Test_Count] , axis=1)
```

```
Train_Count.plot(kind='barh')
```

```
Test_Count.plot(kind='barh')
```

```
plt.style.use('default')
plt.figure(figsize = (25, 8))
image_count = 1
BASE_URL = '/content/train/'

for directory in os.listdir(BASE_URL):
    if directory[0] != '.':
        for i, file in enumerate(os.listdir(BASE_URL + directory)):
            if i == 1:
                break
            else:
                fig = plt.subplot(1, 7, image_count)
                image_count += 1
                image = cv2.imread(BASE_URL + directory + '/' + file)
                plt.imshow(image)
                plt.title(directory, fontsize = 20)
```

RESNET MODEL:

```
# specifing new image shape for resnet
img_shape = 224
batch_size = 64
train_data_path = '/content/train/'
test_data_path = '/content/test/'
train_preprocessor = ImageDataGenerator(
        rescale = 1 / 255.,
        rotation_range=10,
        zoom_range=0.2,
        width_shift_range=0.1,
        height_shift_range=0.1,
        horizontal_flip=True,
        fill_mode='nearest',
    )


test_preprocessor = ImageDataGenerator(
    rescale = 1 / 255.,
)

train_data = train_preprocessor.flow_from_directory(
    train_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size,
    subset='training',
)

test_data = test_preprocessor.flow_from_directory(
    test_data_path,
    class_mode="categorical",
    target_size=(img_shape,img_shape),
    color_mode="rgb",
    shuffle=False,
    batch_size=batch_size,
)
```

```
train_data.class_indices
class_names = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad',
'surprise']
# 224,224,3



ResNet50V2 = tf.keras.applications.ResNet50V2(input_shape=(224, 224, 3)
```

```
                                                    include_top= False,
                                                    weights='imagenet'
                                                    )
#ResNet50V2.summary()
# Freezing all layers except last 50

ResNet50V2.trainable = True

for layer in ResNet50V2.layers[:-50]:
    layer.trainable = False
```

```
def Create_ResNet50V2_Model():

    model = Sequential([
                        ResNet50V2,
                        Dropout(.25),
                        BatchNormalization(),
                        Flatten(),
                        Dense(64, activation='relu'),
                        BatchNormalization(),
                        Dropout(.5),
                        Dense(7,activation='softmax')
                       ])
    return model
```

```
ResNet50V2_Model = Create_ResNet50V2_Model()

ResNet50V2_Model.summary()

ResNet50V2_Model.compile(optimizer='adam',
loss='categorical_crossentropy', metrics=['accuracy'])
```

```
# Create Callback Checkpoint
checkpoint_path = "ResNet50V2_Model_Checkpoint"

Checkpoint = ModelCheckpoint(checkpoint_path, monitor="val_accuracy",
save_best_only=True)

# Create Early Stopping Callback to monitor the accuracy
Early_Stopping = EarlyStopping(monitor = 'val_accuracy', patience = 7,
restore_best_weights = True, verbose=1)

# Create ReduceLROnPlateau Callback to reduce overfitting by decreasing
learning
Reducing_LR = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss',
                                                   factor=0.2,
                                                   patience=2,
#                                                    min_lr=0.00005,
                                                   verbose=1)

callbacks = [Early_Stopping, Reducing_LR]
```

```
steps_per_epoch = train_data.n // train_data.batch_size
validation_steps = test_data.n // test_data.batch_size
```

```
ResNet50V2_history = ResNet50V2_Model.fit(train_data ,validation_data =
test_data , epochs=50, batch_size=batch_size,
                                          callbacks = callbacks,
steps_per_epoch=steps_per_epoch, validation_steps=validation_steps)
```

FLASK app.py :

```
from flask import Flask, render_template, request
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
import numpy as np
from PIL import Image

app = Flask(__name__)
model = load_model('MODELS/mood_resnet_model.h5', compile=False)

@app.route('/')
def index():
    return render_template('index.html')



import pandas as pd
import random

def recommend_song(predicted_mood):
    moosik = pd.read_csv("data_moods.csv")
    moosik = moosik[['name', 'artist', 'mood', 'popularity', 'album', 'id']]
```

```
    RAND_SWITCH_THRESH = 0.15
    p = random.random()


    if((predicted_mood=='happy' or predicted_mood=='sad') and p >=
RAND_SWITCH_THRESH):

        Play = moosik[moosik['mood'] == 'Happy']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        randidx = random.randint(0, 4)
        print(randidx)
        song = Play.iloc[randidx]


    if((predicted_mood=='fear' or predicted_mood=='angry') and p >=
RAND_SWITCH_THRESH):

        Play = moosik[moosik['mood'] =='Calm']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        randidx = random.randint(0, 4)
        song = Play.iloc[randidx]

    if((predicted_mood=='surprise' or predicted_mood=='neutral') and p >=
RAND_SWITCH_THRESH):

        Play = moosik[moosik['mood'] =='Energetic']
```

```
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        randidx = random.randint(0, 4)
        song = Play.iloc[randidx]

    else:
        Play = moosik[moosik['mood'] =='Sad']
        Play = Play.sort_values(by="popularity", ascending=False)
        Play = Play[:5].reset_index(drop=True)
        randidx = random.randint(0, 4)
        song = Play.iloc[randidx]

    return song


@app.route('/upload', methods=['POST'])
def upload():
    if 'image' in request.files:
        image_file = request.files['image']

        # Convert PNG image to JPEG
        img = Image.open(image_file)

        # Convert PNG image to JPEG
        if img.mode != 'RGB':
            img = img.convert('RGB')
        # img = image.load_img(image_file, target_size=(224, 224))
```

```
        img = img.resize((224, 224))
        img_array = image.img_to_array(img)
        img_array = np.expand_dims(img_array, axis=0)



        prediction = model.predict(img_array)
        predicted_class = np.argmax(prediction)

        # Get the class label based on your model's classes
        # Replace 'class1', 'class2', etc. with your actual class labels
        class_labels = ['angry', 'disgust', 'fear', 'happy', 'neutral', 'sad', 'surprise']
        predicted_label = class_labels[predicted_class]
        song = recommend_song(predicted_label)
        song_name = song['name']
        song_artist = song['artist']
        song_album = song['album']
        song_id = song['id']

        return render_template('display.html', song_name = song_name,
song_album = song_album, song_artist = song_artist, song_id = song_id)


    else:
        return 'No image uploaded.'

if __name__ == '__main__':
    app.run()
```

index.html :

```
<!DOCTYPE html>
<html>
<head>
   <link href='https://fonts.googleapis.com/css?family=Martian Mono' rel='stylesheet'>
   <title>Upload Image</title>
   <style>
      body {

         font-family: 'Martian Mono';
         display: flex;
         flex-direction: column;
         justify-content: flex-start;
         align-items: center;
         height: 100vh;
         background-image: url("{{ url_for('static', filename='indeximage.jpg') }}");
         background-repeat: no-repeat;
         background-position: center;
         background-size: cover;
         background-attachment: fixed;

      }
      .container {
         background-color: rgba(255, 255, 255, 0.3);
         padding: 20px;
         border-radius: 10px;
```

```
    text-align: center;
    margin-top: auto;
    margin-bottom: auto;
}

h1 {
    color: #333333;
    font-size: 28px;
    margin-bottom: 10px;
}

input[type="file"] {
    margin-bottom: 10px;
}

input[type="submit"] {
    background-color: #4CAF50;
    color: #ffffff;
    padding: 10px 20px;
    border: none;
    border-radius: 4px;
    cursor: pointer;
}

input[type="submit"]:hover {
    background-color: #45a049;
}
```

```
        .title {
            font-size: 24px;
            margin-top: 10px;
            color: #333333;
            text-align: center;
        }
    </style>
</head>
<body>
    <div class="title">
        <h1>MUSIC RECOMMENDATION SYSTEM BASED ON
MOOD</h1>
    </div>
    <div class="container">
        <h1>Upload face image</h1>
        <form action="/upload" method="post" enctype="multipart/form-data">
            <input type="file" name="image">
            <br>
            <input type="submit" value="Upload">
        </form>
    </div>
</body>
</html>
```

Display.html :

```
 <!DOCTYPE html>
```

```
<html>
<head>
   <link href='https://fonts.googleapis.com/css?family=Calligraffitti' rel='stylesheet'>
   <title>Recommended Song</title>
   <style>
      body {
         font-family: 'Calligraffitti';
         font-size: 22px;
         background-color: #f5f5f5;
         background-image: url("{{ url_for('static', filename='displayimg.jpg') }}");
         background-repeat: no-repeat;
         background-position: center;
         background-size: cover;
         background-attachment: fixed;
      }
      .container {
         max-width: 600px;
         margin: 320px auto;
         background-color: rgba(210, 104, 86, 0);
         padding: 40px;
         border-radius: 10px;
         margin-left: 500px; /*to move the container to the left */
         position: relative;
      }

      h1 {
```

```
    color: rgba(236, 100, 75, 0.8);
    font-size: 50px;
    margin-bottom: 20px;
    text-shadow: 0 0 10px rgba(236, 100, 75, 1);
    animation: flicker 1s linear infinite;
}

h3 {
    color: rgba(236, 100, 75, 0.5);
    font-size: 30px;
    margin-bottom: 10px;
    text-shadow: 0 0 10px rgba(236, 100, 75, 0.8);
    animation: flicker 1s linear infinite;
}

@keyframes flicker {
    0% {
        opacity: 1;
    }
    50% {
        opacity: 0.5;
    }
    100% {
        opacity: 1;
    }
}
```

```
        .wireframe {
            position: absolute;
            top: 0;
            left: 0;
            right: 0;
            bottom: 0;
            border: 3px solid rgba(236, 100, 75, 0.8);
            border-radius: 10px;
            pointer-events: none;
        }

        .glow-behind {
            position: absolute;
            top: -15px;
            left: -15px;
            right: -15px;
            bottom: -15px;
            background: radial-gradient(rgba(236, 100, 75, 0.2), rgba(236, 100, 75,
0));
            border-radius: 10px;
            pointer-events: none;
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Recommended Song</h1>
        <h3>Song Name: {{ song_name }}</h3>
```

```
        <h3>Song Album: {{ song_album }}</h3>
        <h3>Song Artist: {{ song_artist }}</h3>
        <h3>Song Spotify link: <a href="https://open.spotify.com/track/{{ song_id }}">Listen on Spotify</a></h3>
    </div>
</body>
</html>
```

4. Conclusion and Future Work

4.1 Result, Conclusion, and Inference

The project involves using a ResNetVGG50 model to predict the emotions of an image dataset. The predicted emotion is then used to suggest a suitable song. The entire process is implemented on a web app created using Flask and HTML. The potential success of the project relies on a diverse and representative image dataset, fine-tuning the model, and providing a seamless user experience.

Overall, this project combines deep learning, web development, and music recommendation to create a web app that predicts emotions from images

and suggests songs accordingly. It has the potential to provide an engaging and personalized experience for users interested in exploring the emotional aspect of images through music.

4.2 Future Scope

The following future scopes aim to enhance the accuracy, personalization, and real-time capabilities of the project, providing a more comprehensive and immersive user experience while expanding its applications to various domains.

1. Improved Emotion Recognition Models: Explore and develop more advanced deep learning architectures, such as transformer-based models, to improve the accuracy and performance of emotion recognition from images.

2. Multi-modal Emotion Analysis: Extend the project to analyze emotions using multiple modalities, such as text, audio, or facial expressions, to provide a more comprehensive understanding of emotions and enhance the accuracy of predictions.

3. Real-time Emotion Recognition: Adapt the system to perform real-time emotion recognition, allowing for instant analysis of emotions from live video streams or webcam feeds, enabling applications in real-time feedback, video analysis, and interactive experiences.

4. Personalized Music Recommendation: Incorporate user feedback and preference learning algorithms to personalize the song recommendation system, considering individual user preferences, listening history, and emotional responses for more accurate and tailored suggestions.

5. Emotion-aware Content Generation: Explore techniques for generating content, such as images, videos, or text, based on predicted emotions, enabling the creation of emotionally engaging and personalized content for users.

5. Research and writing

5.1 Literature survey

1. Paper: "Deep Emotion Recognition from Facial Expressions Using Convolutional Neural Networks" by J. M. Cohn et al.

i. Advantages: The paper proposes a CNN-based approach for emotion recognition from facial expressions, achieving high accuracy and robustness across different datasets.

ii. Methodology: The authors train a deep CNN model using labeled facial expression images and employ transfer learning to improve performance.

iii. Limitation: The study primarily focuses on facial expressions and does not consider other modalities, such as text or audio, for a more comprehensive emotion analysis.

2. Paper: "Music Recommendation System Based on Emotion Analysis Using Deep Learning" by S. Lee et al.

i. Advantages: This paper presents a music recommendation system that incorporates deep learning techniques for emotion analysis, resulting in more accurate and personalized song suggestions.

ii. Methodology: The authors propose a model that combines CNN and recurrent neural networks (RNNs) to analyze both acoustic features and lyrics of songs for emotion-based recommendation.

iii. Limitation: The study mainly focuses on music analysis and recommendation, without exploring the integration of emotion recognition from images or other modalities.

3. Paper: "Real-Time Emotion Recognition from Images and Videos Using Deep Convolutional Neural Networks" by A. Agarwal et al.

i. Advantages: This paper introduces a real-time emotion recognition system that utilizes deep CNNs to analyze emotions from images and videos, providing instantaneous emotion analysis capabilities.

ii. Methodology: The authors propose a deep CNN architecture trained on labeled emotion datasets, and they optimize the model to achieve low-latency performance for real-time inference.

iii. Limitation: The study focuses on real-time emotion recognition from images and videos, without considering other modalities or incorporating music recommendation based on the predicted emotions.

5.2 REFERENCE

1. Cambridge Spark Blog: "Deploying a Machine Learning Model to the Web"

URL: https://blog.cambridgespark.com/deploying-a-machine-learning-model-to-the-web-725688b851c7

2. Towards Data Science: "Deploying an Image Classification Web App with Python"

URL: https://towardsdatascience.com/deploying-an-image-classification-web-app-with-python-3753c46bb79

3. Towards Data Science: "Build Your First Mood-based Music Recommendation System in Python"

URL: https://towardsdatascience.com/build-your-first-mood-based-music-recommendation-system-in-python-26a427308d96

4. Deploy Machine Learning Model using Flask

https://www.geeksforgeeks.org/deploy-machine-learning-model-using-flask/